Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Смирнов Михаил Александрович

Выпускная квалификационная работа

Автоматическая настройка параметров тиринга, зависящая от входящей нагрузки, в системе хранения данных

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
к.т.н. Лазарева С. В.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Smirnov Mikhail

Automatic setting of tiering parameters depending on workload to data storage system

Graduation Project

Scientific supervisor:
Professor Andrey Terekhov

Reviewer:
Candidate of Engineering Sciences Svetlana Lazareva

Saint-Petersburg
2017

Оглавление

Введение

В последнее время, учёными и исследователями разрабатываются и совершенствуются технологии, предназначенные для замены обычных накопителей на базе магнитных дисков (HDD) [5]. Флэш-память ярко выделяется на фоне остальных и сейчас уже стала один из наиболее популярных способов хранения информации в мобильных устройствах, персональных компьютерах, корпоративных серверах, благодаря высокой скорости доступа, низком энергопотреблении, высокой ударопрочности и портативности. Появление твердотельных накопителей (SSD), оказало значительное влияние на существующую иерархию в системах хранения данных (СХД). От организации архитектуры такой системы будет зависеть её производительность, затраты памяти, срок службы и множество других факторов [2].

Для увеличения производительности и обеспечения сохранности информации систем хранения данных используется технология RAID, объединяющая множество дисков в один логический массив [13]. Самым простым случаем данной технологии является RAID 0, не обеспечивающая сохранности информации при отказе диска, зато отсутствует избыточность при хранении информации. Следующие уровни позволяют предотвращать потерю данных, но с потерей скорости работы или с большими финансовыми затратами из-за избыточности.

Информацию, с которой работают различные СХД, можно условно разделить на "горячую" и "холодную", основываясь на предположении, будет ли она необходима, и как часто будут к ней обращения в будущем. Первостепенная задача в этой области – умение идентифицировать "горячие" данные [9]. Правильность решения проблемы идентификации напрямую будет влиять на производительность всей системы. Если поместить SSD между основной памятью (RAM) и HDD, то флэш-память будет энергонезависимым кэшом второго уровня. В свою очередь, ускорение системы будет достигаться перемещением "горячих" данных на SSD или на SSD, обладающим поддержкой спецификации NVM Express, а "холодных" на HDD, соответственно [1]. Получившая-

ся многоуровневая система хранения данных называется тир (от англ. tiered storage – многоуровневое хранение). Уровней иерархии может быть сколько угодно, но, как правило, используется два или три.

Бакалаврская работа выполнялась в системе RAIDIX. Raidix – программное обеспечение для систем хранения данных, разработанное компанией ООО "Рэйдикс" [10]. Данное решение обладает необходимым функционалом для создания и дальнейшего обеспечения работы механизма тиринга: позволяет создавать RAID-массивы, объединять их в тома, подключать плагины, позволяющие работать с системой хранения данных. Поддерживается параллельный доступ к информации, совместная работа над данными.

1. Постановка задачи

Целью квалификационной работы является исследование и реализация модуля тира [15, 7] в системе RAIDIX, управляющего распределением данных на устройствах хранения. Данное решение должно повысить производительность системы и скорость доступа к критическим данным. Для достижения цели были поставлены следующие задачи, представленные ниже.

1. Анализ существующих алгоритмов организации хранения, выбор оптимального и его улучшение.

2. Реализация поддержки двух и трёх уровней тира, в зависимости от требований пользователя.

3. Функциональное тестирование и тестирование производительности.

2. Обзор

Как уже говорилось выше, самым важным аспектом в реализации тира является правильная идентификация "горячих" данных. Два основных фактора, позволяющих отнести информацию к определённому типу – частота доступа и время обращения. Все алгоритмы идентификации опираются на эти факторы.

2.1. Алгоритмы идентификации

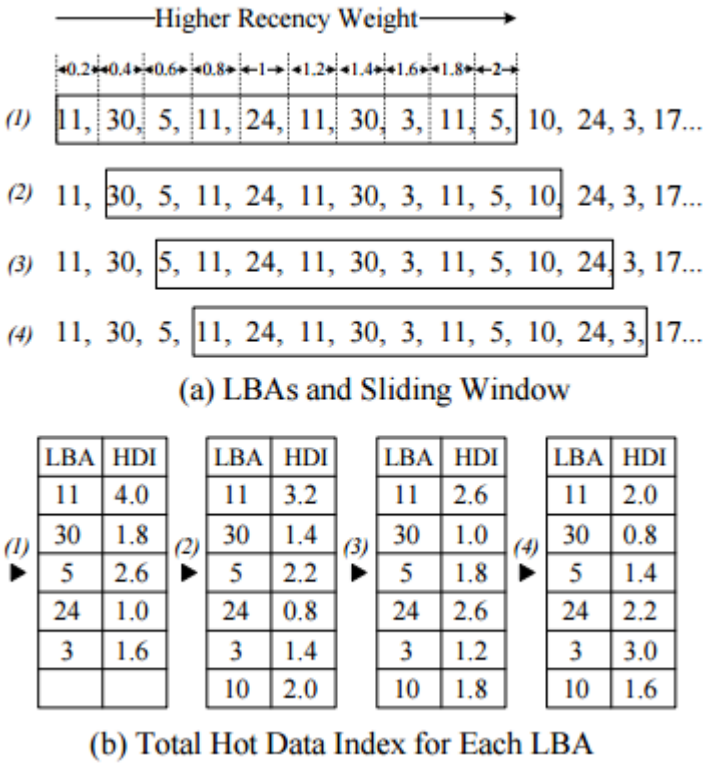

Рис. 1: Пример работы WDAC алгоритма

2.1.1. Window-based Direct Address Counting

В алгоритме [9] анализируется поток данных и, в зависимости от времени запроса, данным присваивается вес. Это можно представить в виде «окна», которое перемещается по потоку (см. Рис. 1). Если один и тот же LBA (англ. Logical block addressing — механизм адресации и

доступа к блоку данных) встречается не один раз, то веса суммируются. Результат работы алгоритма можно представить в виде таблицы. Для идентификации ”горячих” данных необходимо определить некоторое пороговое значение, преодоление которого будет говорить о «горячести» информации.

Для понимания следующих алгоритмов требуется ввести понятие Bloom Filter (см. Рис. 2). Для его работы требуется битовый массив и произвольное количество хэш-функций, в том числе и одна. Изначально битовый массив заполнен нулями. На вход хэш-функций подаётся значение (на практике LBA), и индекс в битовом массиве, соответствующий значению хэш-функции, становится равным единице.

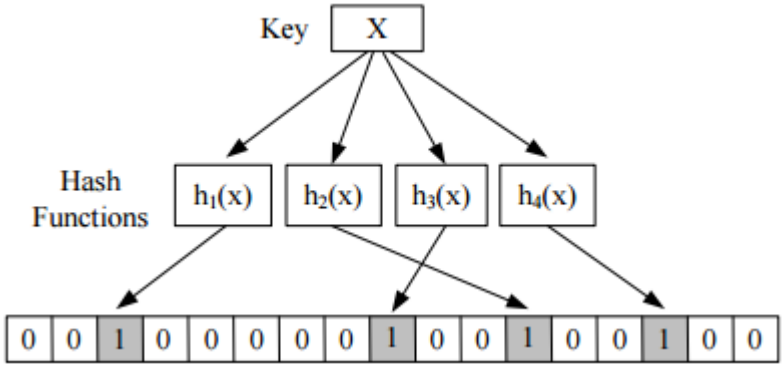

Рис. 2: Bloom Filter

2.1.2. Multiple Hash Function

В алгоритме [4] имеется Bloom Filter, роль битового массива выполняет массив чисел, значения которых являются счётчиком. Когда приходит запрос значение в массиве увеличивается на единицу. Для распределения данных необходимо определить условие, выполнение которого будет говорить о том ”горячие” данные или нет, и на какой уровень в системе хранения данные нужно переносить. Например, если значения в массиве, соответствующие значениям хэш-функций, больше некоторого числа, то данные “горячие”, и “холодные”, если меньше. Устаревание данных реализуется делением значений в массиве на выбранную или настраиваемую константу. Для проверки является ли

определённая ячейка данных “горячей” необходимо подставить её LBA в хэш-функции и проверить выполняется ли условие.

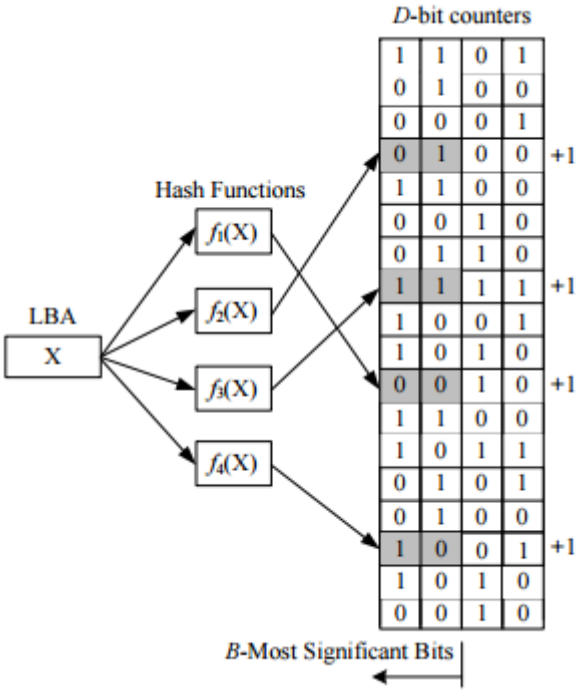

Рис. 3: Multiple Hash Function

2.1.3. HotDataTrap

Алгоритм HotDataTrap (см. Рис. 4) использует только часть всего LBA для работы алгоритма, что экономит пространство памяти [9]. Если LBA 32-х битный, то можно взять последние 16 битов, которые разделим на Primary ID – первые 12 из 16, и Sub ID – оставшиеся 4 бита. В таблице будет храниться информация о том, сколько раз приходил запрос для каждой ячейки и информацию о том, был ли запрос в последнее время. Для устаревания данных можно устанавливать значение Rec в ноль, или делить счётчик доступа на произвольную константу (см. Рис. 4).

Алгоритм Window-based Direct Address Counting не подходит, потому что статистика хранится по количеству запросов, а для тиринга важно анализировать статистику именно за определённый промежуток времени. Алгоритмы HotDataTrap и Multiple Hash Function объединяет

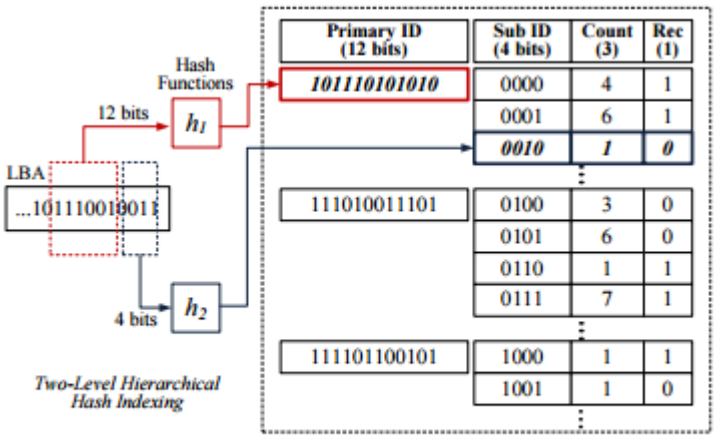

Figure 3.17: HotDataTrap Framework

Рис. 4: HotDataTrap

существование ошибок идентификации, когда ”холодные” данные считаются ”горячими”, но не наоборот. Плюсы – это высокая эффективность и низкое потребление памяти (см. Рис. 5). Но эти алгоритмы различаются принципами снижения объёмов хранимой информации, для RAIDIX больше подходит Multiple Hash Function.

	Фиксированный объём памяти для хранения	Экономие памяти	Ошибки распределения
Window-based Direct Address Counting	+	-	-
Multiple Hash Function	-	+	+
HotDataTrap	-	+	+

Рис. 5: Сравнение алгоритмов идентификации

2.2. Существующие решения

2.2.1. Easy Tier в IBM SVC

Механизм тиринга в продукте IBM SVC получил название Easy Tier [6]. Технология обеспечивает автоматическую миграцию данных между тирами: данные "погорячее" перемещаются на тир с быстрыми носителями, "остывшие" данные – на медленный тир. Easy Tier мониторит ввод-вывод и задержки (I/O activity and latency) при обращении к данным за период 24 часа. На основе анализа промониторенных данных формируется план миграции, после чего данные начинают перемещаться между тирами динамически в зависимости от нагрузки. Easy Tier работает на уровне блоков. Соответственно, размер блока данных, с которым работает авто-тиринг определяется заданным размером экстента (16 – 8182 МБ). Easy Tier можно включить на уровне storage pool или volume. Решение поддерживает до 3х тиров: tier-0 (ssd), tier-1 (enterprise hdd), tier-2 (nearline hdd).

2.2.2. EMC Fast VP

Решение проблемы многоуровневого хранение от корпорации EMC получило название FAST VP [11, 3]. Миграция представляет собой процесс, с помощью которого система перемещает данные на другой уровень и балансирует блоки информации в пределах уровня для достижения максимального прироста производительности. Миграция данных может быть запланирована или ручная. Например, Fast VP позволяет настраивать продолжительность и периодичность миграции. Можно каждый день начинать миграцию в 22:00, когда клиентская активность минимальна, и закончить в 6:00, когда пользователи начинают активней работать с системой. Также существует четыре политики распределения данных, по-разному анализирующие потоки ввода/вывода, мощности дисковых накопителей и объёмы свободного пространства. Размер блока данных равен 256 МБ.

2.3. Linux Kernel Module Programming

Проект реализован в виде модуля ядра Linux. Модули позволяют расширять функциональность ядра, без необходимости его перезагрузки. Основой любой системы хранения данных является параллельный доступ к данным и их блокировка при необходимости. Для этого использовались средства Linux: readlock – установка блокировки на секцию, позволяющая только чтение, writelock – соответственно на запись.

2.4. Средства тестирования

Для тестирования использовалась утилита fio [12], позволяющая создавать сложные сценарии тестирования.

Параметры указываемые при конфигурации сценария (пример см. Listing. 1):

- rw – режим нагрузки: случайное чтение, случайная запись и их комбинация;
- runtime – продолжительность тестирования;
- blocksize – размер блока;
- ioengine – метод доступа к диску;
- size и offset – выделение определённой области диска для работы;
- numjobs – количество клонов потока, выполняющих одну и ту же рабочую нагрузку
- другие параметры;

В процессе и в конце работы утилита fio показывает количество операций в секунду и задержку выполнения операций. Fio позволяет игнорировать файловую систему и обращаться напрямую к диску. Также использовалась утилита dd [8], для отправки единовременных запросов к дискам, и команда iostat для проверки процентной загруженности дисков в системе.

Listing 1: Пример сценария тестирования

```
direct=1
runtime=1H
filename=/dev/tier
ioengine=libaio
iodepth=32
blocksize=4K
write_iops_log=test_log_iops
rw=randread
size=100G
```

3. Реализация двухуровневого и трёхуровневого тира

Для работы с тиром требуется создать на сервере том, который и будет исходным тиром. Для этого создаются RAID массивы для каждого уровня тира. Например, для трёхуровневого необходимо создать RAID с HDD дисками, с SSD дисками и с SSD дисками, обладающим поддержкой спецификации NVM Express (см. Рис. 6).

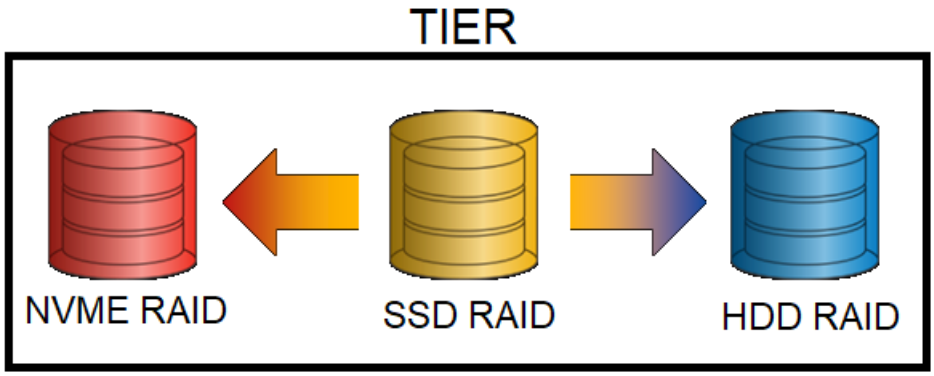

Рис. 6: Архитектура тира

Реализация тира:

1. Язык: C.
2. Среда: Ядро Linux 3.10.0-327 x86_64.
3. Функциональность:
   (a) Собственное логирование.
   (b) Анализ статистики.
   (c) Перенос данных по уровням хранения.
4. Особенности:
   (a) Возможен выбор двух или трёх уровней системы хранения.
   (b) Возможно тестирование в промышленной системе RAIDIX.

Для реализации одной из версий рабочего тира было решено упростить часть, связанную с идентификацией. Для простоты каждой ячейке памяти соответствует значение в массиве, величина которого равна количеству обращений к этой ячейке. Когда необходимо начать переносить информацию между уровнями тира массивы “холодных” и “горячих” данных сортировались по частоте доступа. “Холодные” – по убыванию, “горячие” – по возрастанию. Ячейки, с наибольшей частотой доступа среди “холодных”, меняются местами с ячейками, с наименьшей частотой среди “горячих”.

Миграция данных представляет собой процесс, при котором значения в ячейках данных меняются местами. Для переноса двух блоков памяти создаются 2 запроса на запись в ячейки, затем генерируются 2 запроса на чтение (см. Рис. 7). И чтение и запись происходит небольшими блоками фиксированного размера. После того, как процессы чтения закончат работу, потоки на запись изменят значения ячеек. Процесс продолжается до тех пор пока блоки данных целиком не поменяются местами. Ячейки памяти могут быть размером от 1 МБ до 1 ГБ, а

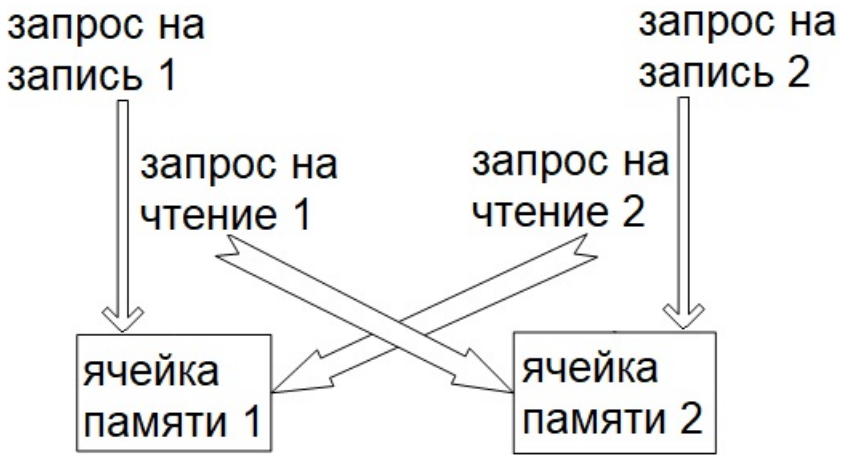

Рис. 7: Схематическое изображение процесса миграции двух ячеек

миграция данных может идти с абсолютно любой периодичностью. В случае если процесс миграции не закончился, а уже должна по времени начаться следующая, то текущая останавливается, идёт перерасчёт статистики, создаётся план новой миграции и запускается новое перераспределение между уровнями.

4. Алгоритм идентификации ”горячих” данных

После реализации трёхуровневого тира, я встроил в систему алгоритм Multiple Hash Function, который позволил оптимизировать процесс миграции – для каждой ячейки памяти можно сразу определить ”горячая” она или нет без необходимости анализа всей статистики и перенести ячейку на соответствующий ей уровень.

Процент горячих данных необходимо поддерживать равным процентному соотношению флэш-памяти ко всему объёму хранилища. Для этого перед миграцией проверяется процент “горячих” данных, если он слишком большой, то повышаем уровень ”горячести” данных, если наоборот, то уменьшаем.

Для полноценной работы системы хранения необходимо выделять неактуальные данные, для этого после миграции ”устареваем” данные делением их статистики на 2, при этом статистика данных, к которым не было обращения с момента прошлой миграции, делится на 3, для выделения наименее актуальной информации.

Алгоритм Multiple Hash Function имеет следующие настраиваемые параметры:

1. Количество хэш-функций.
2. Размер Bloom Filter.
3. Размер элементов Bloom Filter.
4. Размер ячейки памяти.

Для выбора оптимальных значений параметров, от которых зависит эффективность работы алгоритма, я провёл серию функциональных и исследовательских тестов для каждого из параметров и затем алгоритма в целом.

5. Функциональное тестирование и измерение производительности

Для тестирования использовались две системы.
Параметры первой системы:

- 10 HDD дисков, суммарным объёмом 700 ГБ, объёдинённые в RAID-массив;
- 3 SSD диска, суммарным объёмом 335 ГБ, объёдинённые в RAID-массив;
- RAID-массивы объединены в один том, который является тиром;
- кэш для каждого из RAID-массивов выделен в размере 5 ГБ;
- ядро – Linux 3.10.0-327 x86_64 SMP;
- процессор – Intel(R) Xeon(R) CPU E5620 @ 2.40GHz;

Вторая система аналогично первой, только добавляется дополнительный уровень, состоящий из NVME дисков, суммарным объёмом 2 ТБ.

Первичная проверка корректности переноса производилась с помощью программы dd. В системе отправлялись запросы на чтения и запись в определённые ячейки, затем запускалась миграция. После миграции проверялся адрес ячеек (LBA - Logical block addressing) и проверялась корректность переадресации. Для этого, с помощью команды md5sum [14], считались хэш-суммы данных до распределения и сравнивались с результатом вычисления после миграции. В результате, данные успешно переносились между уровнями.

5.1. Схема экспериментов

Общая схема тестирования:

1. Моделируется нагрузка и запускается тестирование производительности RAIDIX.
2. Запускается механизм распределения данных.
3. Через 1 час или более тестирование приостанавливается и снимаются показатели производительности.

Характеристики производительности:

1. Количество операций ввода/вывода в секунду (IOPS - input/output operations per second).
2. Объём перенесённых данных.
3. Другие параметры.

5.2. Тестирование корректности алгоритма

Сценарий тестирования:

- продолжительность – 1 час;
- размер нагружаемой области – 100 ГБ;
- периодичность миграции – 1 минута;
- размер блока памяти – 256 МБ;
- количество хэш-функций – 2;
- размер Bloom Filter – 8000;

Скорость работы сильно возросла под конец теста (см. Рис. 8), при этом нагрузка на HDD диски была равна нулю, а на SSD диски выше 99%.

Затем был запущен аналогичный тест, но длительностью 24 часа и с периодом миграции 1 час (см. Рис. 9). На рисунке видно, что когда началась первая миграция, через час, производительность начала быстро возрастать, достигла какого-то предела и продолжала оставаться в

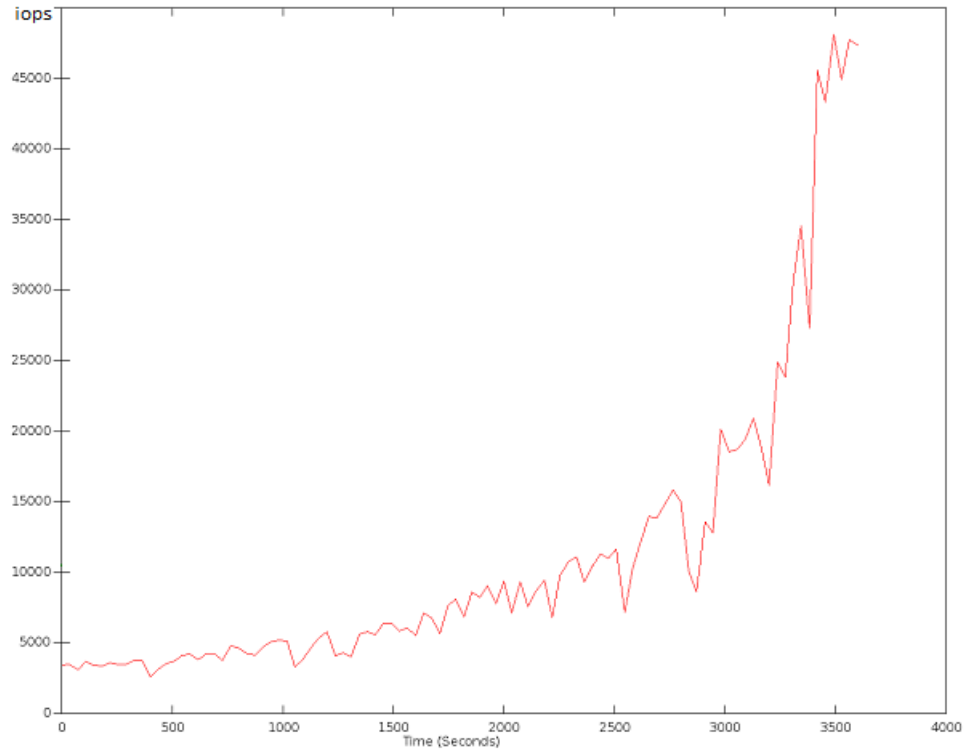

Рис. 8: Показатели IOPS в тесте на корректность работы

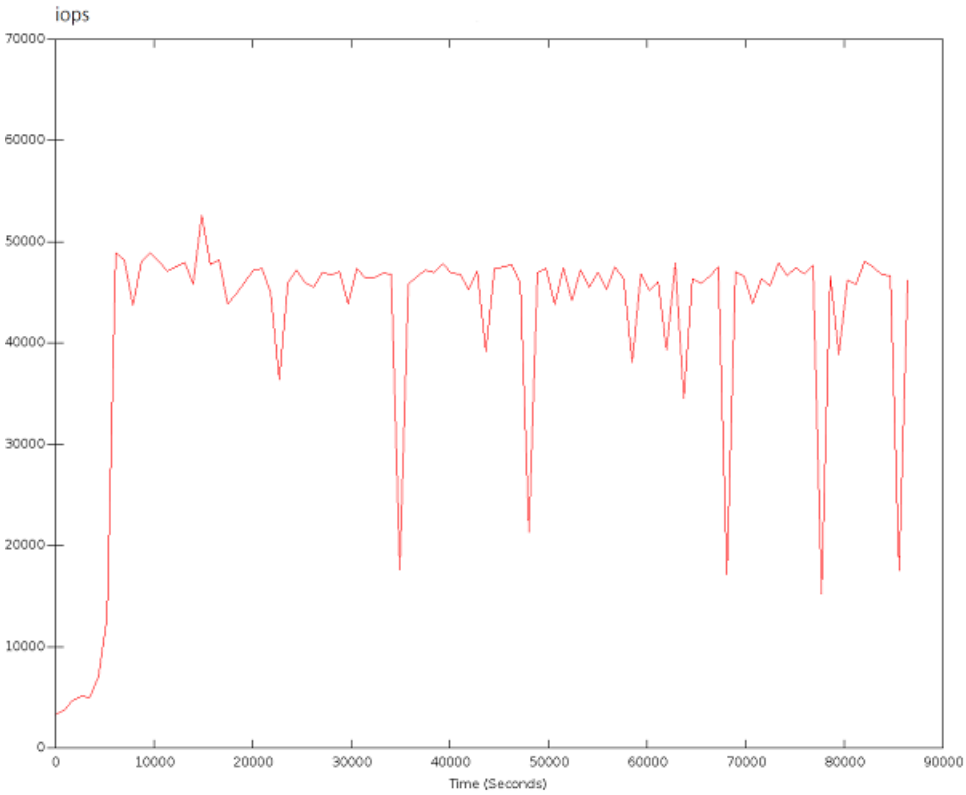

Рис. 9: Показатели IOPS в длительном тесте на корректность работы

некотором диапазоне значений. На графике видны провалы производительности, которые могут быть вызваны двумя причинами: первая – запрос на чтение попал в область, которая, в данный момент, находится в состоянии миграции и пришлось ждать окончания данного процесса, вторая – износ ssd-диском. Для проверки второй причины было проведено тестирование SSD-RAID и SSD дисков (см. Пункт 5.2.).

5.3. Тест производительности SSD-RAID

Для проверки SSD дисков, вначале нагружался случайным чтением весь SSD-RAID, а затем каждый из трёх дисков по-отдельности. Произ-

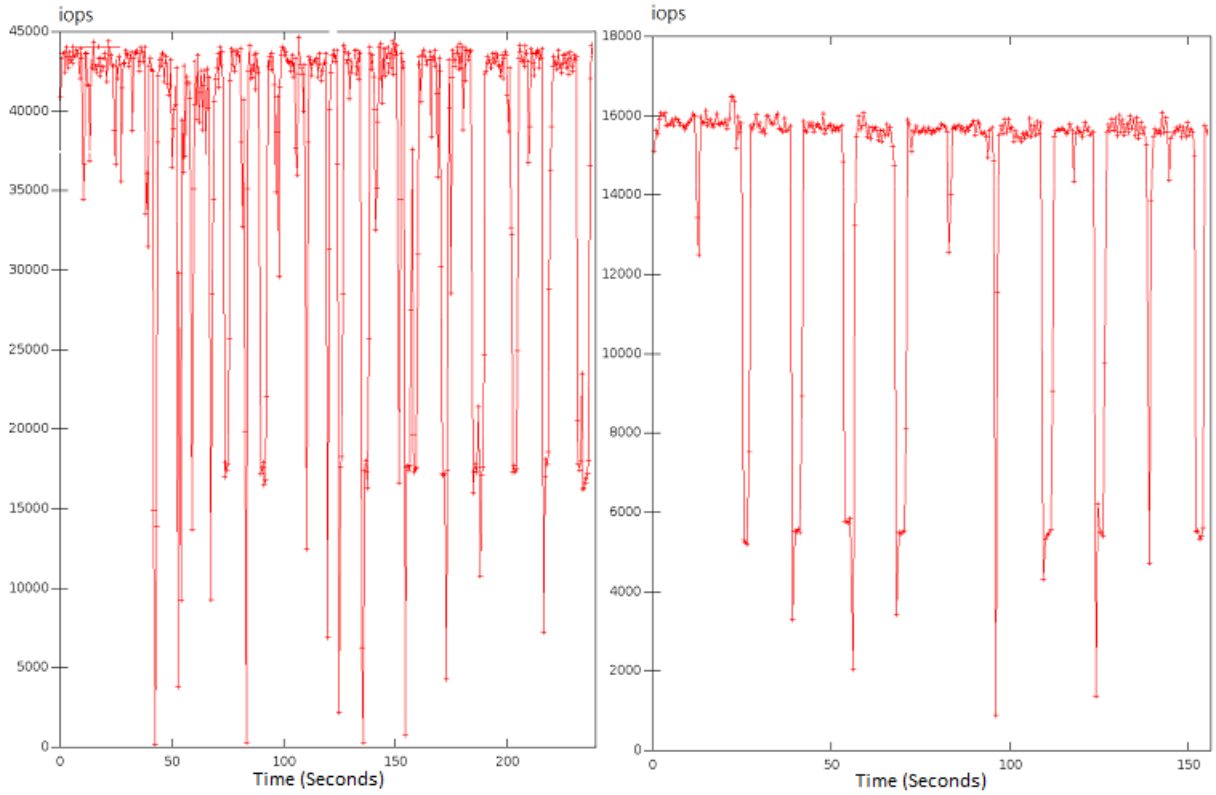

Рис. 10: Показатели IOPS для SSD-RAID (слева) и SSD дисков (справа)

водительность SSD-RAID-а и SSD дисков периодически падает почти до нуля (см. Рис. 10), что объясняет провалы в длительном тесте на корректность (см. Рис. 9). Самая вероятная причина возникновения таких провалов – износ дисков.

5.4. Тестирование параметров тира

5.4.1. Тест типа данных Bloom Filter

Выбор типа данных для Bloom Filter заключался в выборе между atomic или atomic64. Atomic - это 32-bit переменная, но первые 8 бит - это lock, оставшиеся 24 - signed 24-bit integer, т.е. ограничено сверху значением 8 388 608. Верхняя граница atomic64 превышает миллиард. Предположим, что все мощности системы направлены в одину ячейку памяти, это максимально критическая ситуация. Если идёт работа с HDD-RAID из 10 дисков, который выдаёт 3 000 операций в секунду, то понадобится примерно час, чтобы произошло переполнение. Если идёт работа с SDD-RAID из 3 дисков, который выдаёт 40 000 IOPS, то понадобится всего 4 минуты, чтобы произошло переполнение. Конечно, это критические ситуации и на практике такое случается редко. Рассмотрим более реальную модель нагрузки – нагружается случайным чтением область 100 ГБ на SSD-RAID, так как он намного быстрее HDD-RAID, и посмотрим значения в Bloom Filter. В итоге получаются подобные значения (см. Рис. 11). Нагрузка была достаточно критическая, область относительно небольшая и длился тест 1 час 20 мин, устаревания данных не было, полученное максимальное значение – 3 330 650. Получаем, что в таком режиме тир сможет работать больше трёх часов. Если учесть, что устаревание данных может происходить только раз в 24 часа, то это очень мало времени до переполнения. Поэтому было решено использовать 64-х битную переменную.

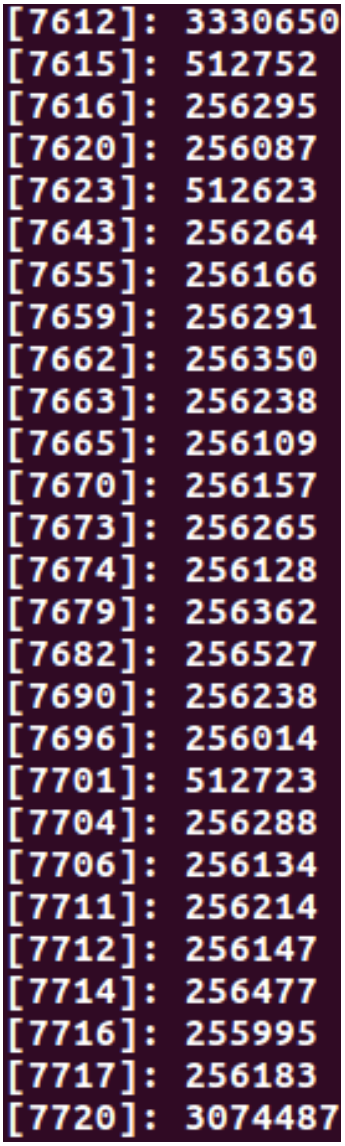

Рис. 11: Часть значений Bloom Filter.

5.4.2. Тест на определение количества хэш-функций

Чтобы понять какое количество хэш-функций использовать была проведена серия тестов, в которых увеличивалось количество хэш-функций в алгоритме. Сценарий тестирования в остальных пунктах совпадает со сценарием тестирования корректности (см. Пункт 5.2). Начиная с этой серии тестов и во всех последующих было "включено" устаревание данных, по аналогии с реальными системами, где без этого невозможно представить себе работу. В качестве хэш-функций была взята стандартная хэш-функция hash_64 [7] и хэш-функции hashRs, hashLy, hashRot13 [16].

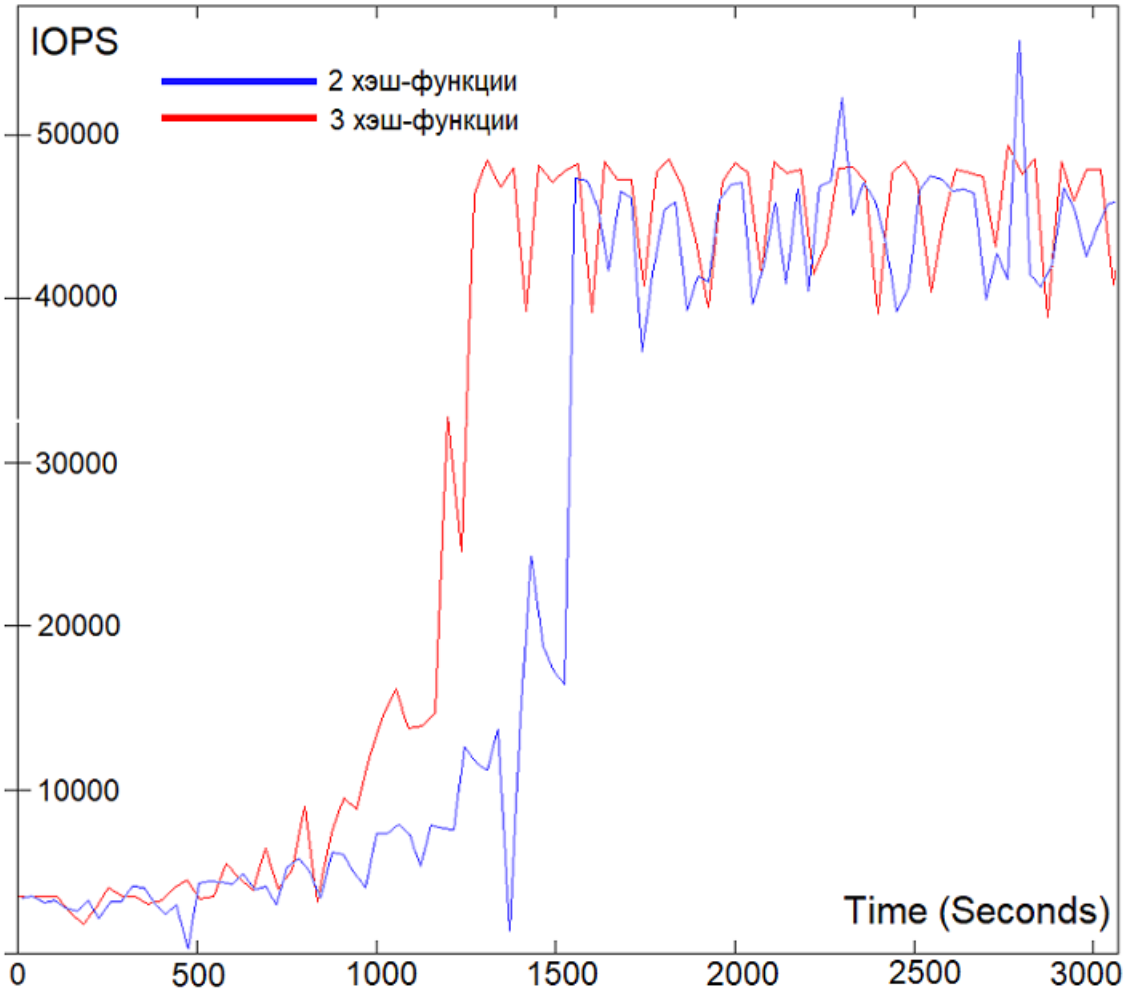

Рис. 12: Показатели IOPS для тестов с двумя и тремя хэш-функциями

В случае двух хэш-функций было мигрировано 140 ГБ данных, в случае трёх 115 ГБ (см. Таблицу 1). Что помимо уменьшения коли-

чества бесполезной работы дало сравнительно быстрый рост производительности. Примерно 2000 сек. против примерно 1300 сек. во втором случае, т.е. 11,5 мин выигрыша (см. Рис. 12).

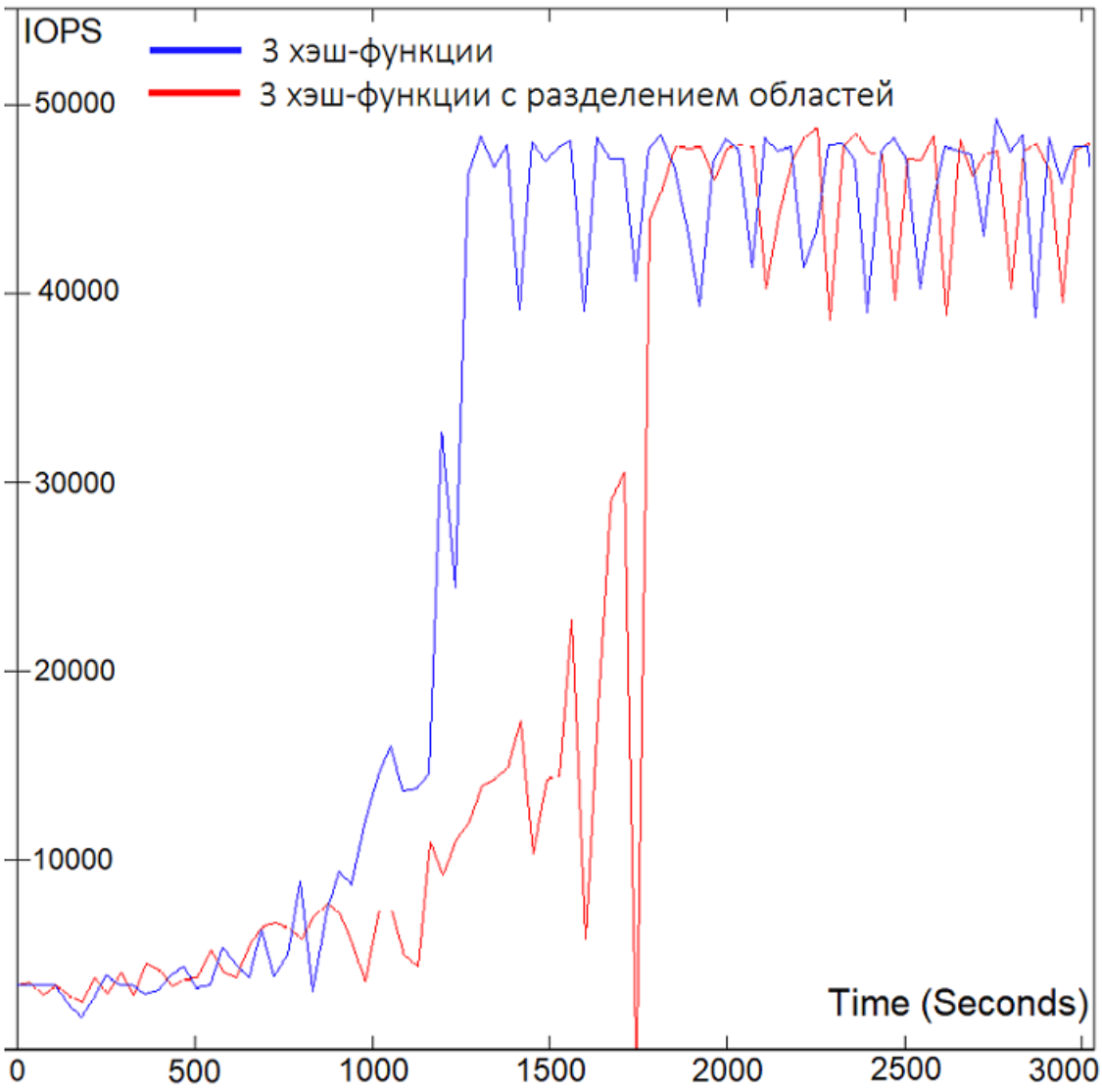

Рис. 13: Показатели IOPS для тестов с тремя хэш-функциями

Затем было принято решение протестировать для трёх хэш-функций, но с раздельными областями для хэширования (см. Рис. 13). В результате было перенесено 130 ГБ данных (см. Таблицу 1), что лучше, чем для двух хэш-функций, но хуже, чем просто для трёх. Было принято решение отказаться от этой идеи.

Для четырёх хэш функций было перенесено 108 ГБ против 115 ГБ у трёх (см. Таблицу 1). Рост производительности также был быстрее. Примерно 1200 сек. против примерно 1300 сек. (см. Рис. 14).

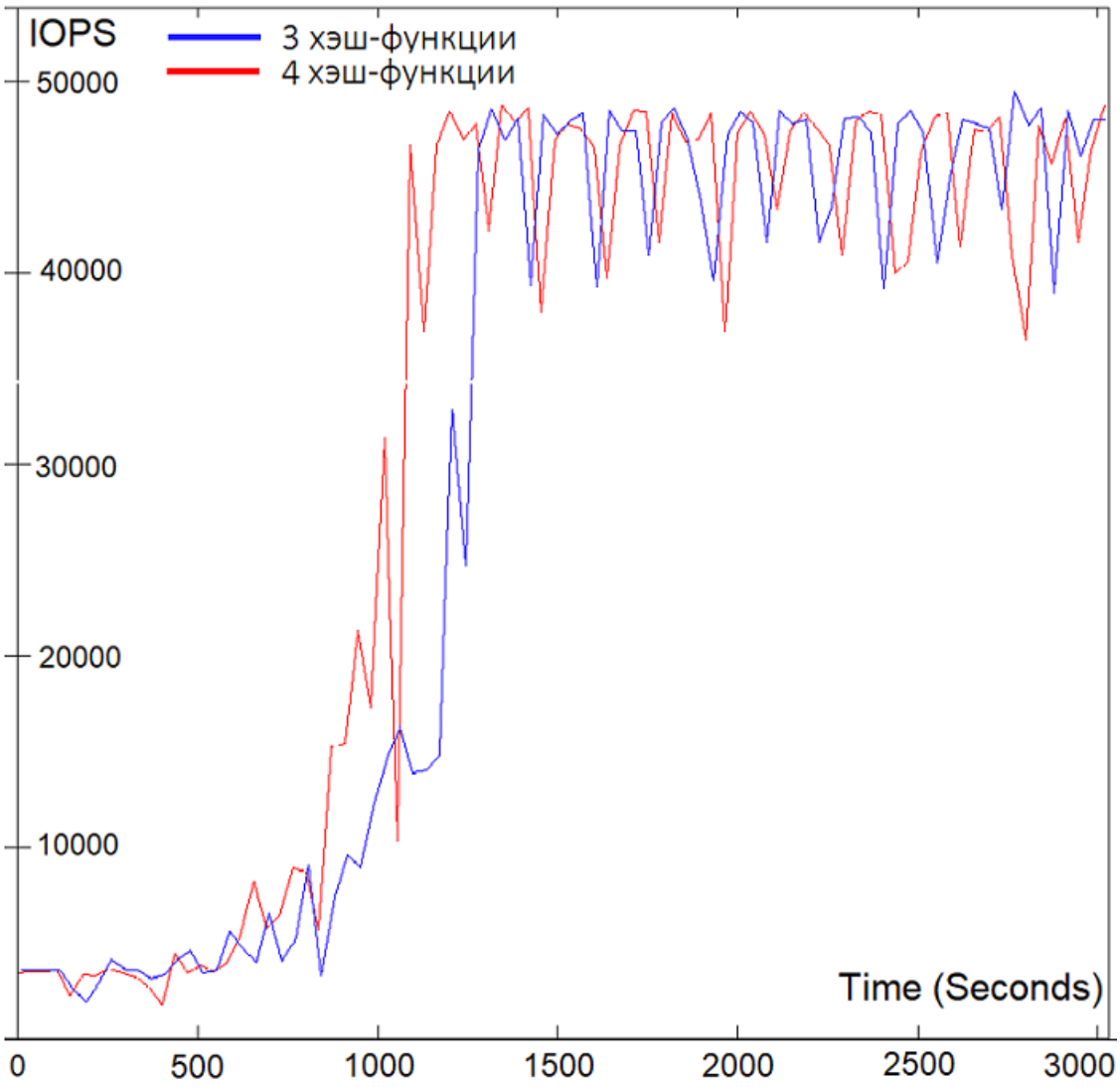

Рис. 14: Показатели IOPS для тестов с тремя и четырьмя хэш-функциями

Таблица 1: Характеристики производительности

Количество хэш-функций	Объём мигрируемых данных, ГБ	Время резкого роста, сек
2	140	$\approx$ 2000
3	115	$\approx$ 1300
3*	130	$\approx$ 1750
4	108	$\approx$ 1200

Для пяти хэш-функций, во-первых, рост производительности будет ещё медленнее, так как для четырёх хэш-функций замедлился в срав-

нении с тремя, а, во-вторых, чем больше хэш-функций, тем больше значений в Bloom Filter и, при больших объёмах хранилищ и большой нагрузке, будет наблюдаться обратная картина - чем больше хэш-функций, тем больше ошибок.

5.4.3. Тесты на определение размера Bloom Filter

Для выяснения оптимального размера Bloom Filter проводим серию тестов и в каждом тесте меняем размер Bloom Filter. Сценарий тестирования в остальных пунктах совпадает со сценарием тестирования корректности (см. Пункт 5.2) Ожидаем, что при увеличении размера Bloom Filter, количество "ошибочных" миграций будет уменьшаться.

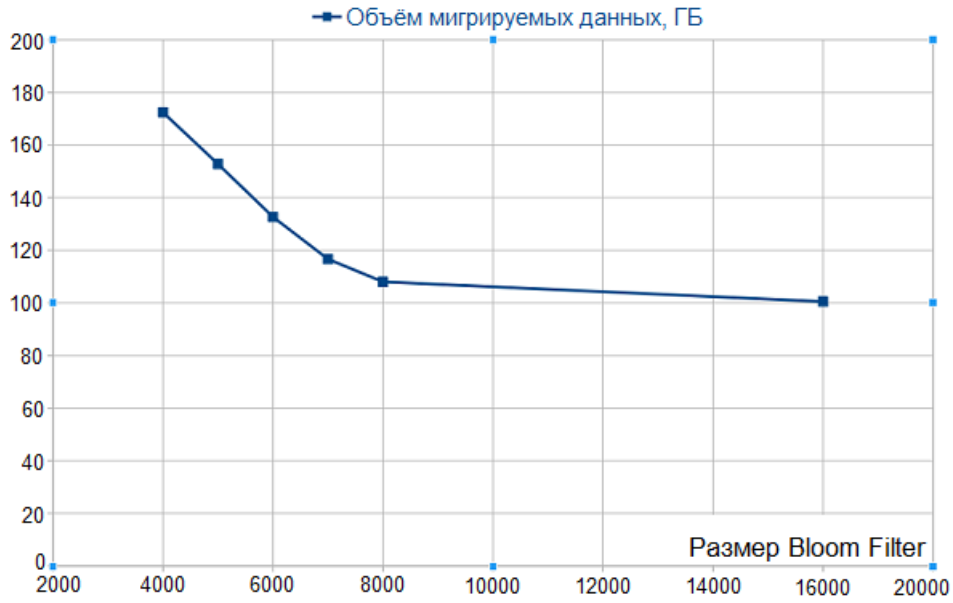

Рис. 15: Объём мигрируемых данных, в зависимости от размера Bloom Filter

Для Bloom Filter размером 4000 было перенесено 173 ГБ данных (см. Рис. 15, 16), для Bloom Filter размером 16000 перенесли 100,5 ГБ информации. Для успешной работы тира приемлемыми являются размеры от 7300, где экономия памяти на статистических данных равна 10%, и погрешность переноса примерно равна 10%.

Размер Bloom Filter	Объём мигрируемых чанков, GB
4000	100,5
5000	108
6000	117
7000	133
8000	153
16000	173

Рис. 16: Соответствие объёма мигрируемых данных и размера Bloom Filter

5.5. Исследование поведения системы при различных типах нагрузки

В следующих тестах стояла задача смоделировать поведение пользователя и исследовать поведение системы при такого типа нагрузках.

5.5.1. Сравнение с тиром без миграции

Один поток нагружает весь тир случайным чтением. Ещё три потока, по очереди в течение 12 часов, нагружают случайным чтением области HDD-RAID, следующие друг за другом.

Сценарий тестирования:

- продолжительность – 36 часов;
- размер нагружаемой области – 100 ГБ;
- периодичность миграции – 1 час;
- размер блока памяти – 256 МБ;
- количество хэш-функций – 4;
- размер Bloom Filter – 8000;

Тот же самый тест запустим позже, но отключим миграцию данных.

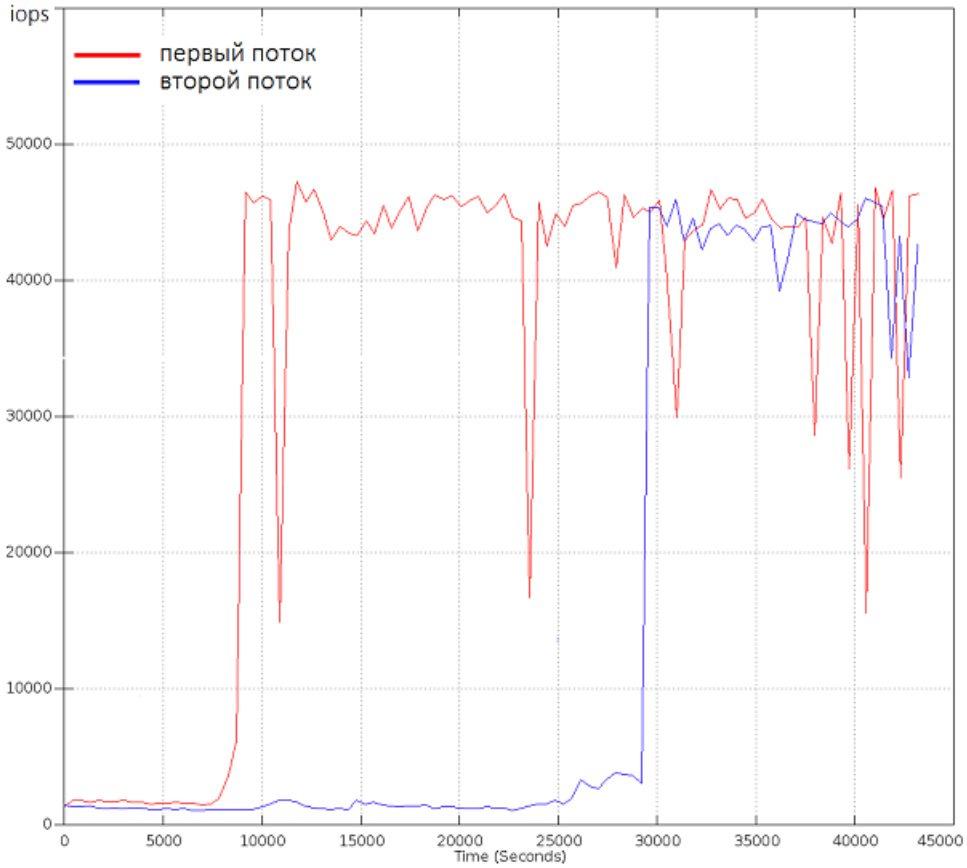

Рис. 17: Показатели IOPS для потоков по двум областям HDD-RAID

Первая область переехала быстрее, но, в итоге, все три потока достигли одинаковой производительности (см. Рис. 17). Третий поток аналогичен второму. На рисунке 18 показано сравнение производительности одних и тех же потоков с миграцией и без. Всего было мигрировано 360 ГБ данных, что много для трёх областей суммарным объёмом 300 ГБ. Учитывая, что размер SSD-RAID 300 ГБ, можно понять как происходил процесс миграции: погрешность для четырёх хэш-функций округлим до 10%. После окончания работы первого потока, на SSD-RAID было занято 110 ГБ, после второго потока 220 ГБ. Третьему потоку необходимо 110 ГБ, поэтому с SSD-RAID вытесняется 30 ГБ данных. Получаем $220 + 110 + 30 = 360$. Погрешность всё так же осталась 10%.

Исходя из полученных результатов (см. Рис. 18) делаем вывод, что система с тиром выгоднее, так как рост производительности до 1800% окупает затраты на миграцию.

Параметр сравнения	С миграцией	Без миграции	С миграцией лучше?
Средняя скорость потока по всему тиру, iops	2300	1800	+
Средняя скорость первого потока по области, iops	34000	1800	+
Средняя скорость второго и последующих потоков по области, iops	15000	1800	+
Объём мигрируемых данных, ГБ	360	0	-

Рис. 18: Сравнение показателей системы с миграцией и без

5.5.2. Тестирование миграции при нагрузке на SSD-RAID

Для тестирования миграции при сильной нагрузке на SSD-RAID, был создан сценарий тестирования, в котором один поток всё время нагружал случайным чтением область размером на SSD-RAID, а другие потоки по очереди в течение восьми часов нагружали области на HDD-RAID.

Сценарий тестирования:

- продолжительность – 48 часов;
- размер нагружаемой области – 100 ГБ;
- периодичность миграции – 1 час;
- размер блока памяти – 256 МБ;
- количество хэш-функций – 4;
- размер Bloom Filter – 8000;

Изначально поток на SSD-RAID выдавал более 45 000 IOPS, когда данные начали переноситься с HDD-RAID, показатели IOPS начали падать и остановились на 25 000 (см. Рис. 20). Поток, изначально нагружающий область на HDD-RAID, остановился примерно там же. Мощность,

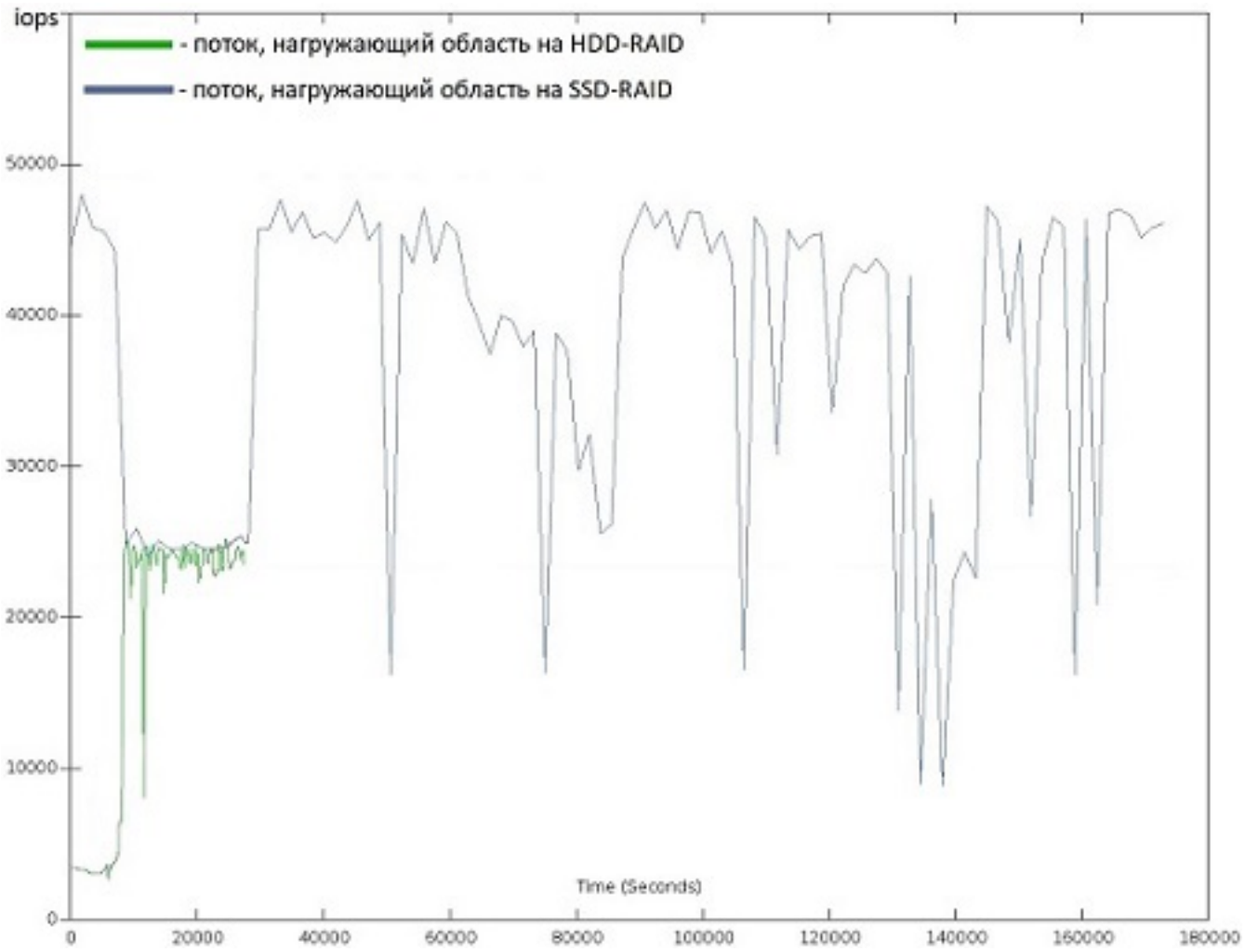

Рис. 19: Показатели IOPS для потока на SSD-RAID и потока на HDD-RAID

по сути, разделилась между ними. Графики остальный потоков не приведены, так как они аналогичны первому потоку, который нагружает область на HDD-RAID.

5.6. Объём метаданных

С использованием восьми битной переменной для хранения данных (см. Пункт 5.3.1), получается такая зависимость количества метаданных от размера ячейки памяти (см. Рис. 20). Например, для ячейки памяти объёмом 4 МБ необходимо 5,5 ГБ метаданных на 1 ТБ данных, когда для ячейки объёмом 1 ГБ необходимо 22 МБ метаданных на 1 ТБ информации. Если увеличивается размер ячейки памяти, то

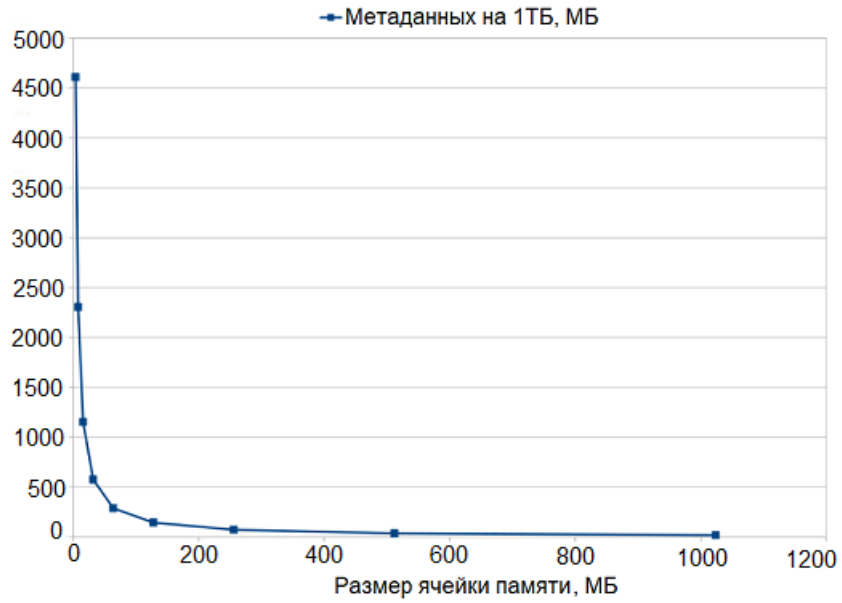

Рис. 20: Зависимость метаданных от размера ячейки памяти

уменьшается производительность. Больше неактуальной информации переносится – ”горячие” данные весят 100 МБ, но механизм перенесёт 1 ГБ. Ещё возможны попадания в блокированные данные – тир меняет местами два блока памяти и блокирует их для чтения и записи, но приходит запрос от пользователя на эти данные и ему приходится ждать окончания переноса данных. Чем меньше ячейка памяти тем быстрее закончится перенос и тем меньше вероятность попадания запроса в мигрирующие области.

Заключение

В рамках данной работы была исследована и реализована многоуровневая система хранения данных – тир. Были выполнены следующие задачи:

- проведён анализ алгоритмов распределения данных и выбран алгоритм наилучшим образом удовлетворяющий потребностям СХД RAIDIX;
- реализован прототип модуля, организующий миграцию данных между уровнями хранения;
- проведено тестирование производительности с использованием синтетической нагрузки;
- на основании тестирования произведён выбор ключевых параметров алгоритма;

Список литературы

[1] Chen Peter P. S. Optimal file allocation in multi-level storage systems. — Harvard University Cambridge, Massachusetts, 1973. — URL: http://dl.acm.org/citation.cfm?id=1499662.

[2] EMC. Information storage and management: Storing, Managing, and Protecting Digital Information. / Ed. by EMC Education Services G. Somasundaram, Alok Shrivastava. — Wiley Publishing, Inc.

[3] EMC Corporation. Managing FAST VP // Unity Family technical documentation. — URL: https://www.emc.com/ru-ru/documentation/unity-family/unity-p-pool-config/05-unity-pool-br-using-fast-vp.htm (online; accessed: 17.05.2017).

[4] Jen-Wei Hsieh Tei-Wei Kuo Li-Pin Chang. Efficient Identification of Hot Data for Flash Memory Storage Systems. — ACM Transactions on Storage, 2006.

[5] Jeong-Uk Kanga Jin-Soo Kima Chanik Parkb Hyoungjun Parkb Joonwon Leea. A multi-channel architecture for high-performance NAND flash-based storage system. — 2007. — URL: http://dl.acm.org/citation.cfm?id=1244544.

[6] Kushal S. Patel Shrikant V. Karve. IBM Easy Tier. — IBM Systems and Technology Group, 2014. — URL: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/9dc4420fb6fcd0b386257f0200559e17/\protect\char'0024\relaxFILE/Easy%20Tier%20for%20SVC%20and%20V7000%20whitepaper.pdf.

[7] Linux Cross Reference. — URL: http://lxr.free-electrons.com/ (дата обращения: 18.12.2016).

[8] Linux по-русски. Изучаем команды Linux: dd // Виртуальная энциклопедия. — URL: http://rus-linux.net/ (дата обращения: 19.12.2016).

[9] Park Dongchul. Hot and Cold Data Identification: Applications to Storage Devices and Systems. — 2012.

[10] RAIDIX. Программно-определяемая система хранения данных. — URL: http://www.raidix.ru/ (дата обращения: 15.04.2017).

[11] White Paper. EMC VNX2 FAST VP. — EMC Corporation, 2016. — URL: https://www.emc.com/collateral/white-papers/h12102-vnx-fast-vp-wp.pdf.

[12] fio - Linux man page // Виртуальная библиотека. — URL: https://linux.die.net/man/1/fio (дата обращения: 10.04.2017).

[13] В.Г. Казаков С.А. Федосин. Технологии и алгоритмы резервного копирования. — Мордовский государственный университет им. Н.П. Огарева, 2008. — URL: http://window.edu.ru/resource/176/56176.

[14] Мир Gnu / Linux // Ресурс для начинающих пользователей. — URL: http://linux-user.ru/komandy-v-linux/podschity-vaem-md5-hesh-summu-v-linux/ (дата обращения: 10.01.2017).

[15] Олег Цилюрик. Программирование модулей ядра Linux. — 2007.

[16] Хабрахабр. — URL: https://habrahabr.ru/post/219139/ (дата обращения: 12.01.2017).